METHOD AND APPARATUS FOR COHERENT MEMORY STRUCTURE OF HETEROGENEOUS PROCESSOR SYSTEMS

TECHNICAL FIELD

The invention relates to a coherent multiprocessor (MP) bus for communication between caches used by non-homogeneous processors.

BACKGROUND

One method of extending or adding to the processing capability of a processing unit is to offload some of the work or functions to one or more coprocessors. Typically one processor is a main or control plane processor which is used to coordinate the work of the remaining data plane processors. Each of these processors will typically have their own cache memory. Data plane processors will also typically have their own Local Memory. As is known, caches are essentially temporary storage memory directly or closely linked to a processor. The amount of time required to retrieve information from a cache is significantly less than the time required to retrieve information from hard disk or even from RAM (Random Access Memory. For this reason, a given processor, in a multiprocessor system, typically, when needing information not in its associated cache, will perform a request for the information from memory. Other processors in the multiprocessor system will first perform a lookup or "snoop" request to ascertain if the information being requested is contained in the cache associated with that "another processor." If so, the information may be directly retrieved from the other cache and each cache in the system is advised if state changes are required in the cache for that memory location, as set forth infra. Thus caches, even of other processors, are used to reduce latency or the time that it

takes to retrieve information, especially where that information is likely to be used multiple times by a given processor.

To prevent destruction of data being used or accessed simultaneously by different processors and to further assure that data accessed is the most recently correct value, coherent memory systems have been developed and used to communicate the status of data to all processors using data contained in a given virtual or physical memory location. One example of a coherent memory management system uses a protocol referred to in the industry as a MESI protocol. Each read or write request involves all other caches in the system. After each read or write request relative a given memory location, each cache in the system maintains a record of the cache state with respect to that memory location of Invalid, Shared, Exclusive or Modified in accordance with the MESI protocol. More information on such coherency protocols may be obtained from many sources such as the Internet and so forth. An extended MESI protocol outlined in this invention is set forth in a U.S. Patent 6,334,172 issued December 25, 2001 and assigned to IBM. Known prior coherent memory systems were, however, limited to processors executing the same instruction set, having the same cache hierarchy, and each accessing or addressing memory in the same manner.

A prior art multiprocessing system having heterogeneous processors is shown in an application Serial Number 09/736,585 filed December 14, 2000, entitled Symmetric Multi-Processing System and assigned to the same assignee as the present invention. This system is also shown in Publication 2002/0078308 published June 20, 2002. In this system, the main processing units (PUs) can access memory at either a physical or a virtual memory location using Load and Store instructions. However, the auxiliary (APU) or synergistic (SPU) processor units access memory using a DMA (Direct Memory Access) unit and

have a different cache hierarchy than the PUs. Elsewhere, in IBM prior art, memory management schemes have been described for the DMA units to access memory using a physical and/or a virtual address. In addition, the SPUs have a Local Storage which may be addressed as part of system memory. In other words, the PUs and the APUs were heterogeneous or non-homogeneous.

It may be pointed out that in such heterogeneous processor systems, the APUs are typically specialized processors for doing specific jobs more efficiently, and with less hardware than is used in conjunction with the central or control processors.

Typically, homogeneous multiprocessor systems provide instructions for performing atomic updates of a memory location. Atomic updates are useful in cooperative programming environments to perform operations such as "compare and swap," "test and set," "fetch and no-op," and "fetch and store". These instructions typically rely on a coherent access of a system memory location. A means for a DMA unit to issue atomic update of memory has been described in IBM prior art. When combined with the present invention, the SPU can perform the above operations in a compatible manner as the control processor(s) and thus participate in a cooperative programming environment with the control processor.

Further in the IBM prior art, methods are described for allowing a DMA operation to be cached and also for data to be predictably pre-fetched. As is known, caches and pre-fetching data typically are provided in a multiprocessor system to improve the performance of a cooperative programming environment. However, to take advantage of these methods in a shared memory system with the control processor, the cached and pre-fetched data must be coherent with the other system caches.

While the characteristics of a prior art non-homogeneous processor can be implemented without a coherent memory

structure, the performance and difficulty in programming in such an environment suffers greatly. It would thus be highly desirable to have a system whereby each processor, regardless of processor configuration, can time efficiently and in a coherent manner communicate with other caches used by other processors in the system to minimize time required to retrieve valid and up-to-date information utilized by more than one processor of a multiprocessor system.

SUMMARY OF THE INVENTION

The present invention sets forth a method of communication between processors and their associated caches as used by non-homogeneous processors to provide a coherent memory system.

BRIEF DESCRIPTION OF THE DRAWINGS

For a more complete understanding of the present invention, and its advantages, reference will now be made in the following Detailed Description to the accompanying drawings, in which:

FIGURE 1 is a block diagram of a non-homogeneous multiprocessor system;

FIGURE 2 is a flow diagram of the process used in updating cache tables to provide coherency of data; and

Figure 3 is an illustration of the System Memory Map associated with the multiprocessor system of Figure 1.

DETAILED DESCRIPTION

In FIGURE 1, a dash line block 10 includes a first or type A processor unit or PU 12 and an MMU (Memory Management Unit) 14. For the purposes of this discussion, PU 12 may be considered as a general purpose processor operating in a control or central processing unit (CPU) mode. While the PU may comprise many components besides the MMU, such components are not specifically

shown to simplify this explanation of operation of the invention. The MMU 14 operates in conjunction with a cache 16 to communicate via a bus 18 with other devices and processors as well as with a system memory block 20. As known in the art, an MMU operates to translate processor instructions and data effective addresses to physical memory addresses. The block 20 may include not only ROM (Read Only Memory) and RAM (Random Access Memory), but also Memory Mapped devices such as a hard disk for maintaining the virtual memory. The bus 18 is further labeled as a coherent SMP (Symmetric MultiProcessor) processor bus. Another type A processor 22 is shown communicating via a cache 24 with the bus 18. A different configuration device or processor is shown within a dash line block 26. Within block 26 is shown a processor or other device block 28 communicating with a local store 30 and further utilizing an MMU 34 and a DMA block 32 to communicate with other entities via an optional cache 36 and bus 18. For the purposes of this discussion, block 28 will typically be a processor of a different configuration than the type A processor 10. In a preferred embodiment, it will be a special purpose processor which is programmable, however, more suited to a class of applications than the type A general purpose processor. An example of a processor might be a DSP (Digital Signal Processor). Such a processor primarily will typically operate on data inserted into the local storage block 30 by a control processor. Local storage block 30 is mapped into the physical address space of the system and may be directly accessed in the same manner as system memory block 20 by a control processor or indirectly by the DMA controller block 32. On occasion, processor 28 may need to transfer additional information between local storage 30 and system memory 20 using the DMA controller 32. Even though described as a programmable processor, block 28 may however be a single function device such

as a graphics controller or one providing MPEG (Moving Picture Experts Group) compression. Another type B block 38 is shown communicating with bus 18 via the same cache 36 as is used by block 26. A third block 40 is shown labeled as being a type B PU. However the invention is capable of supporting a multiplicity of processors, device configurations, and cache hierarchies that need access to a common system memory. As shown, block 40 communicates with bus 18 via a cache 42, although it could use any other cache capable of communicating with a plurality of devices such as the cache 36. The bus 18 is shown with arrows on each end to illustrate that the bus may be connected to other devices external to or a part of the multiprocessor system.

As is known, the MMUs 14 and 34 use a table stored in system memory for the translation of a virtual address to a real address. In a cooperative programming environment, the MMUs 14 and 34 share a common translation table. When a program requires more memory than available real memory, the operating system moves a region of real memory to a hard disk or other non-directly addressable storage media to make room for the data required by the program at that moment in time. This process is known as "paging." As memory is paged in and out of real memory, the operating system must update the translation table to reflect the new virtual to real mappings. Without coherency, updates of the translation table may not get reflected in the MMUs and future translations may be incorrect. The coherent memory structure put forth in this invention insures that all future translations are correct.

In a cooperative programming environment, the atomic update of system memory is required for synchronization of programs executing on the control and data plane processors. The atomic update of memory typically is a read-modify-write operation

which relies on a coherent memory structure for proper operation. First, an atomic load is performed to read the current data for the memory location. When the load is performed, a reservation is also set for the corresponding memory location. Next, the data is operated on by a program and written back to the same memory location using an atomic store. If another processor or device accesses the data between the atomic load and the atomic store, the reservation is lost and the atomic store will fail. If the atomic store fails, the program performs the atomic read-modify-write operation again.

The flow diagram of FIGURE 2 commences with a request from a processor, or other device, for data contained at a given address in system memory 20. A first step 50 is for the request to be sent to all the other caches in the system that are connected to bus 18. Each of these caches, as set forth in a step 52, look up the address in a table located in each of the caches. The state of all of these caches, as set forth in the protocol of the coherent system being used, are combined and returned to the requestor as well as to all the remaining caches, as set forth in step 54. The table internal to each of the caches is then updated to the next state of the cache line indicated by the address involved in the request.

FIGURE 3 illustrates a typical memory map for the non-homogeneous multiprocessor system depicted in FIGURE 1. A Real Address Space 100 represents the mapping of various system resources into the address space of a control processor 200 and a data plane processor 300. The block 100 further includes a system memory block 110, an LS (Local Storage) Alias block 120, an MMU & MMIO registers block 130, an I/O devices block 140 and a system ROM block 150. The processor 200 further includes an MMU block 210 and a cache 220. The specialized or heterogeneous processor 300 is shown having an MMU block 320, a DMA block 330,

a local storage block 340 and an optional cache block 350. System memory 110 represents the DRAM and SRAM storage of the system. MMU and MMIO registers block 130 represents the mapping of the Memory Management translation tables, the control registers for the DMA controller 330 and Memory Management Unit (MMU) 320, as well as other system control registers. I/O Devices area 140 represents the mapping of I/O devices, such as a Hard Disk controller, and the System ROM area 150 represents the mapping of non-volatile memory. Local Storage Alias 120 represents the mapping of the Local Storage blocks 340 into the real address space of the system. The Local Storage Alias allows other processors within the non-homogeneous multiprocessor to directly access data stored in memory of a data plane processor such as 340.

Mapping the Local Storage of a data plane processor into the real address space of the system allows a control processor, another data plane processor, or other system device to directly access data in Local Storage. Since the Local Storage looks like system memory, these accesses may be cached and thus require a coherency mechanism to prevent the destruction of the data.

As will be realized by the above, the present invention comprises providing access to system memory in a coherent memory managed manner to heterogeneous processors and other devices. One coherent memory space is provided for a plurality of heterogeneous processors. The coherent access to system memory and single memory space allows the efficiencies of the cache structures to be realized for transfers of data between the local storage and system memory. In other words, the more efficient prior art system of communication of cache to cache transfers of modified or shared data and the direct transfer of data between the system caches and the local storage, such as

optional cache 36, 42 or 350, is not utilized or is bypassed in the present invention.

Although the invention has been described with reference to a specific embodiment, the description is not meant to be construed in a limiting sense. Various modifications of the disclosed embodiment, as well as alternative embodiments of the invention, will become apparent to persons skilled in the art upon reference to the description of the invention. It is therefore contemplated that the claims will cover any such modifications or embodiments that fall within the true scope and spirit of the invention.